# Midterm 1

Yuan Liang

**Problem 1: (total 10 points)**

Short problems:

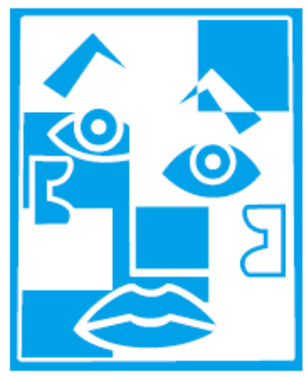
(1) Name 3 of great ideas in designing Computer Architecture according to you and briefly explain what does each of them mean (3 points)

## Design for Moore's Law

The one constant for computer designers is rapid change, which is driven largely by **Moore's Law**. It states that integrated circuit resources double every 18–24 months. Moore's Law resulted from a 1965 prediction of such growth in IC capacity made by Gordon Moore, one of the founders of Intel. As computer designs can take years, the resources available per chip can easily double or quadruple between the start and finish of the project. Like a skeet shooter, computer architects must anticipate where the technology will be when the design finishes rather than design for where it starts. We use an "up and to the right" Moore's Law graph to represent designing for rapid change.


## Use Abstraction to Simplify Design

Both computer architects and programmers had to invent techniques to make themselves more productive, for otherwise design time would lengthen as dramatically as resources grew by Moore's Law. A major productivity technique for hardware and software is to use **abstractions** to represent the design at different levels of representation; lower-level details are hidden to offer a simpler model at higher levels. We'll use the abstract painting icon to represent this second great idea.




## Make the Common Case Fast

Making the **common case fast** will tend to enhance performance better than optimizing the rare case. Ironically, the common case is often simpler than the rare case and hence is often easier to enhance. This common sense advice implies that you know what the common case is, which is only possible with careful experimentation and measurement (see Section 1.6). We use a sports car as the icon for making the common case fast, as the most common trip has one or two passengers, and it's surely easier to make a fast sports car than a fast minivan!

PARALLELISM

## Performance via Parallelism

Since the dawn of computing, computer architects have offered designs that get more performance by performing operations in parallel. We'll see many examples of parallelism in this book. We use multiple jet engines of a plane as our icon for **parallel performance**.

PIPELINING

## Performance via Pipelining

A particular pattern of parallelism is so prevalent in computer architecture that it merits its own name: **pipelining**. For example, before fire engines, a "bucket brigade" would respond to a fire, which many cowboy movies show in response to a dastardly act by the villain. The townsfolk form a human chain to carry a water source to fire, as they could much more quickly move buckets up the chain instead of individuals running back and forth. Our pipeline icon is a sequence of pipes, with each section representing one stage of the pipeline.

PREDICTION

## Performance via Prediction

Following the saying that it can be better to ask for forgiveness than to ask for permission, the final great idea is **prediction**. In some cases it can be faster on average to guess and start working rather than wait until you know for sure, assuming that the mechanism to recover from a misprediction is not too expensive and your prediction is relatively accurate. We use the fortune-teller's crystal ball as our prediction icon.

HIERARCHY

## Hierarchy of Memories

Programmers want memory to be fast, large, and cheap, as memory speed often shapes performance, capacity limits the size of problems that can be solved, and the cost of memory today is often the majority of computer cost. Architects have found that they can address these conflicting demands with a **hierarchy of memories**, with the fastest, smallest, and most expensive memory per bit at the top of the hierarchy and the slowest, largest, and cheapest per bit at the bottom. As we shall see in Chapter 5, caches give the programmer the illusion that main memory is nearly as fast as the top of the hierarchy and nearly as big and cheap as the bottom of the hierarchy. We use a layered triangle icon to represent the memory hierarchy. The shape indicates speed, cost, and size: the closer to the top, the faster and more expensive per bit the memory; the wider the base of the layer, the bigger the memory.


## Dependability via Redundancy

Computers not only need to be fast; they need to be dependable. Since any physical device can fail, we make systems **dependable** by including redundant components that can take over when a failure occurs *and* to help detect failures. We use the tractor-trailer as our icon, since the dual tires on each side of its rear axels allow the truck to continue driving even when one tire fails. (Presumably, the truck driver heads immediately to a repair facility so the flat tire can be fixed, thereby restoring redundancy!)

(2) What is the difference between the add and addu MIPS instructions?

**add is for signed number, addu is for unsigned number.**
**add may cause an overflow exception, while addu will not.**

(3) True/False: circle the correct answer (1 point each)

**T** F 1. Branch instructions in MIPS can only jump forward 32768 and backward 32767 instructions.

a branch with an immediate of 0 jumps forward 1 instruction

T **F** 2. A carry-out at the most significant bit after an addition of two signed numbers always indicates overflow.

operations with a negative result will always have carry-out

**T** F 3. If we only have three parameters to send to a non-recursive function, then we can use registers and don't need to use the stack.

T **F** 4. Every location in the text segment is accessible from a single branch statement.

**Problem 2 (10 points):**

Binary bits have no inherent meaning. Given the bit pattern:

**10110100**

What does it represent, assuming that it is

a. An unsigned integer?
b. Signed magnitude
c. 1's complement
d. 2's complement

a) (2 points)
**180**

b) (2 points)
**-52**

c) (2 points)
**-75**

d) (2 points)
**-76**

We are defining 8-bit floating-point precision, with the following format:

| Sign (1 bit) | Exponent (3 bits) | Fraction (4 bits) |
|---|---|---|

Assuming that it follows the same philosophy as single and double precision defined by IEEE 754 standard.

e) (2 points) What should be the bias for this 3-bit exponent? Leave your answer in decimal.

**3**

f) (2 points) What is the binary representation of the smallest float, which is strictly larger than 1? What are its values in binary and decimal?

**00110001**

**1 + 2^-4 = 1.0625**

**Problem 3 (4 points):**

Assume i and j are assigned to $s0 and $s1. The base address of the array A and B are in registers $s2 and $s3 respectively. Convert the following C code to MIPS code.

C code:

B[7] = A[i] + A[j];

```
sll $t0, $s0, 2     #word to byte conversion i = i*4;
add $t0, $t0, $s2   #address of A[i]
lw  $t2, 0($t0)     #load A[i]
sll $t1, $s1, 2     #j = j*4;
add $t1, $t1, $s2   #address of A[j]
lw  $t3, 0($t1)     #load A[j]
add $t4, $t2, $t3   #temp = A[i]+A[j]
sw  $t4, 28($s1)    #save temp to B[7]
```

**Problem 4 (12 points):**

Consider the application A and the baseline MIPS processor with the following Instruction cycles. Suppose application A executes three billion instructions. Answer the question below, and explain your work.

| Instruction | % of Instructions in *A* | Instruction Latency (cycles) |
|---|---|---|
| Load | 20% | 5 |
| Store | 10% | 4 |
| Simple R-type (i.e. adds, ands, shifts) | 45% | 4 |
| Multiply | 10% | 5 |
| BEQ/BNE | 10% | 3 |
| Jump | 5% | 3 |

***Suppose processor has different clock cycles for different instructions.***

(1) What is the CPI for application A?

**0.2 * 5 + .01 * 4 + 0.45 * 4 + 0.1 * 5 + 0.1 * 3 + 0.05 * 3 = 4.15**

(2) The hardware running the application has a cycle time of 300ps. What is the execution time for that hardware to run the application.

**3 * 10^9 * 4.15 * 300 * 10^-12 = 3.735**

(3) Now suppose we add an instruction that does a multiply/accumulate operation. The multiply/accumulate operation replaces one multiply instruction and one add instruction. This optimization allows the compiler to replace 50% of multipies. What is the new CPI?

**20 / 95 * 5 + 10 / 45 * 4 + 40 / 95 * 4 + 5/ 95 * 5 + 10 / 95 * 3 + 5 / 95 * 3 + 5 / 95 * 7 = 4.27**

(4) What is the execution time for the same hardware from part (2) to run the new application from part (c)?

**0.95 * 3 * 10^9 * 4.27 * 300 * 10^-12 = 3.65**

**Problem 5 (12 points):**

**Carry Look-Ahead adder:**

Given the following truth table for a full adder:

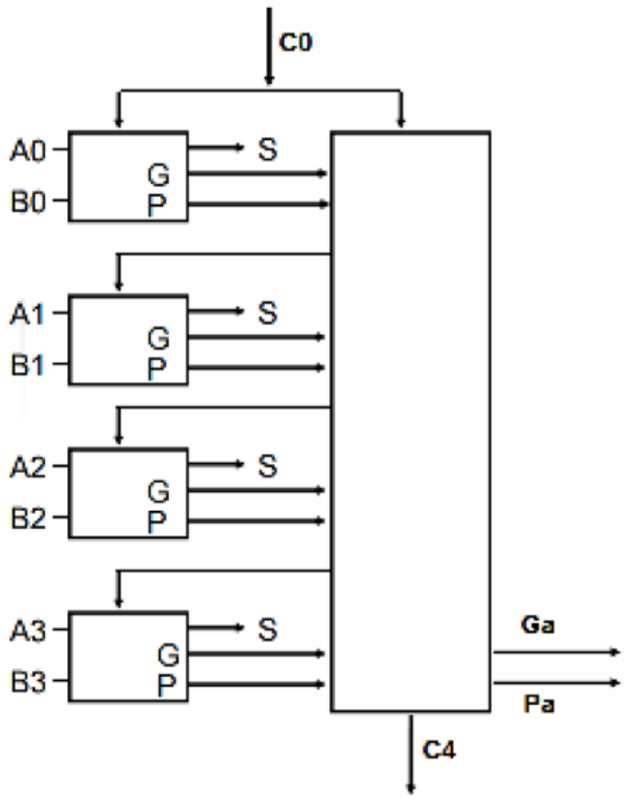
| Inputs | | | Outputs | |
|---|---|---|---|---|
| $A$ | $B$ | $C_{in}$ | $S$ | $C_{out}$ |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

The $G$ (generate) can be defined as $A*B$ and $P$ (propagate) can be defined as $A+B$.

(1) Write out the expression for S and Cout given A, B, and Cin for a 1-bit adder with truth table as shown above.

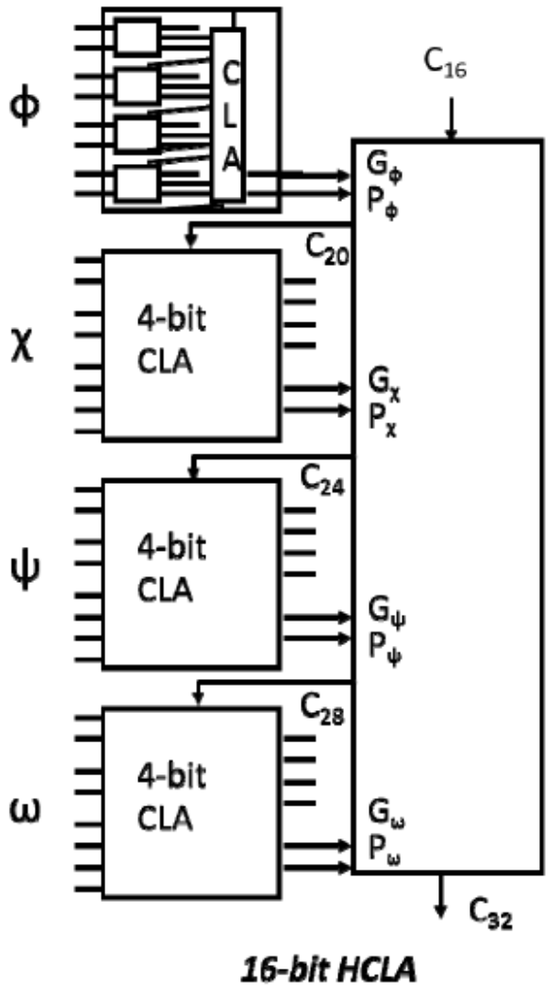
**S= Cin ^ (A ^ B)**
**Cout = (A * B) + (A * Cin) + (B * Cin)**

(2) Given Cout = G + Cin*P. Write out the expression for C4, Ga, and Pa with given C0, Ai, and Bi as shown in figure below, which is a 4-bit Carry Lookahead Adder (CLA).




**Pi = Ai ^ Bi**
**Gi = Ai * Bi**
**Ga = G0*P1*P2*P3 + G1*P2*P3 + G2*P3 + G3**
**Pa = P0*P1*P2*P3**
**C4 = Ga + C0 * Pa**

(2) A 16-bit Hierarchical CLA can be built by four 4-bit CLA's with the way as shown in figure. Write out the expression for C20, C24, C28 and C32, when given $G\phi, G\chi, G\psi, G\omega, P\phi, P\chi, P\psi, P\omega$ and C16.




***16-bit HCLA***

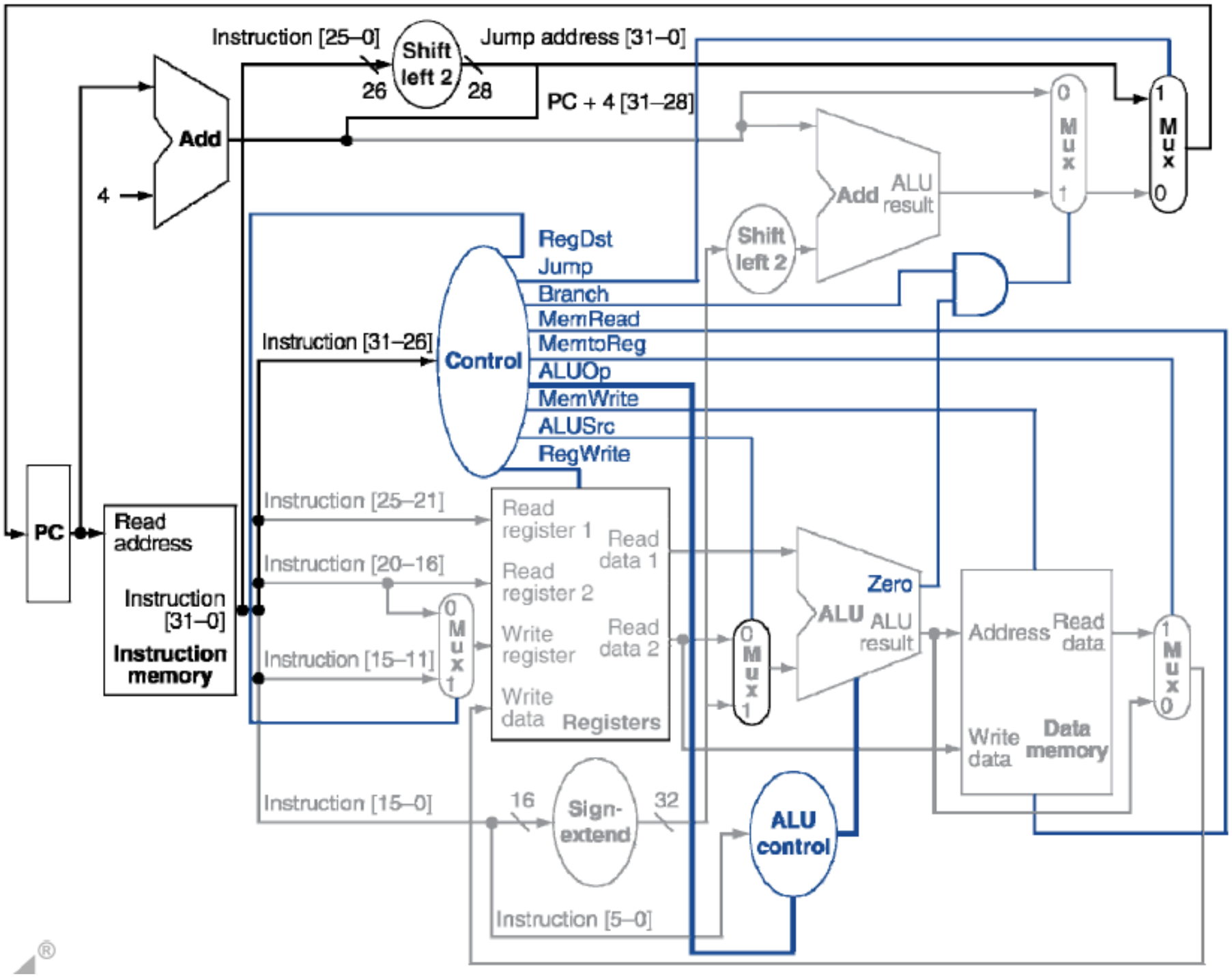
$C20 = G\phi + P\phi \cdot C16$

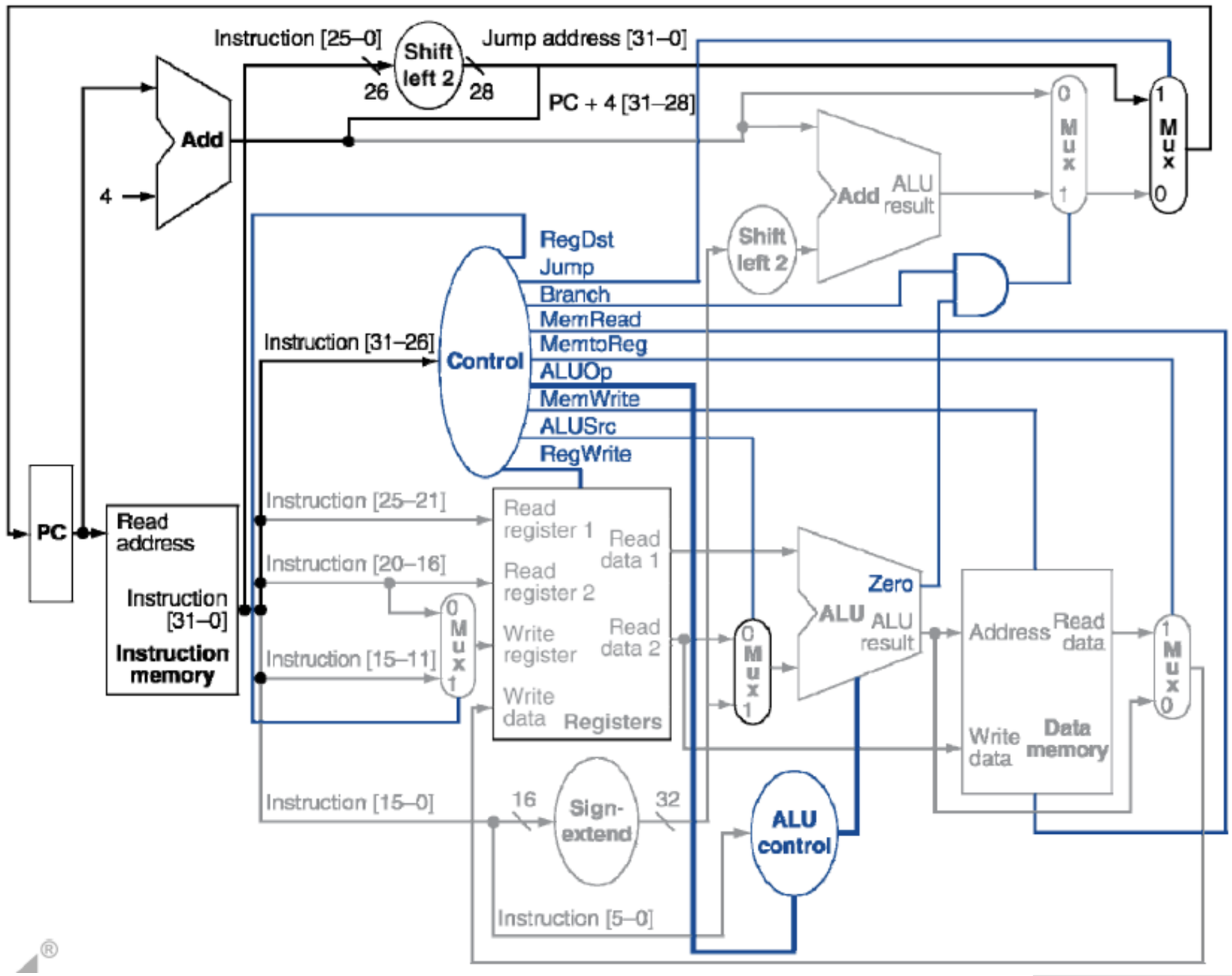
$C24 = Gx + Px \cdot G\phi + Px \cdot P\phi \cdot C16$

$C28 = G\psi + P\psi \cdot Gx + P\psi \cdot Px \cdot G\phi + P\psi \cdot Px \cdot P\phi \cdot C16$

$C32 = Gw + Pw \cdot G\psi + Px \cdot P\psi \cdot Gx + Pw \cdot P\psi \cdot Px \cdot G\phi + Pw \cdot P\phi \cdot Px \cdot P\phi \cdot C16$

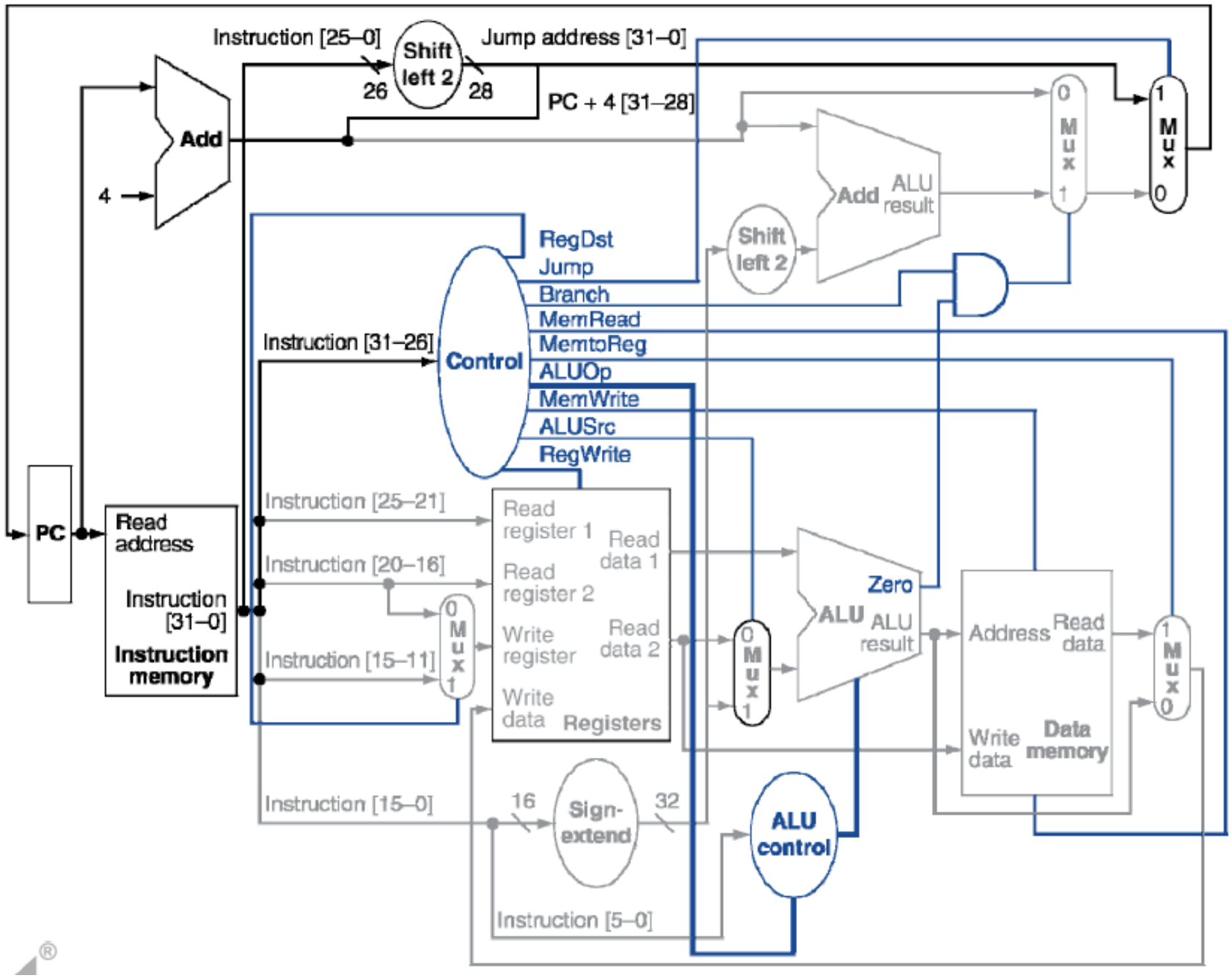
(1) Find the values for all control signals in the single cycle implementation for the following instructions. (fill in 1, 0, or X for does not matter)

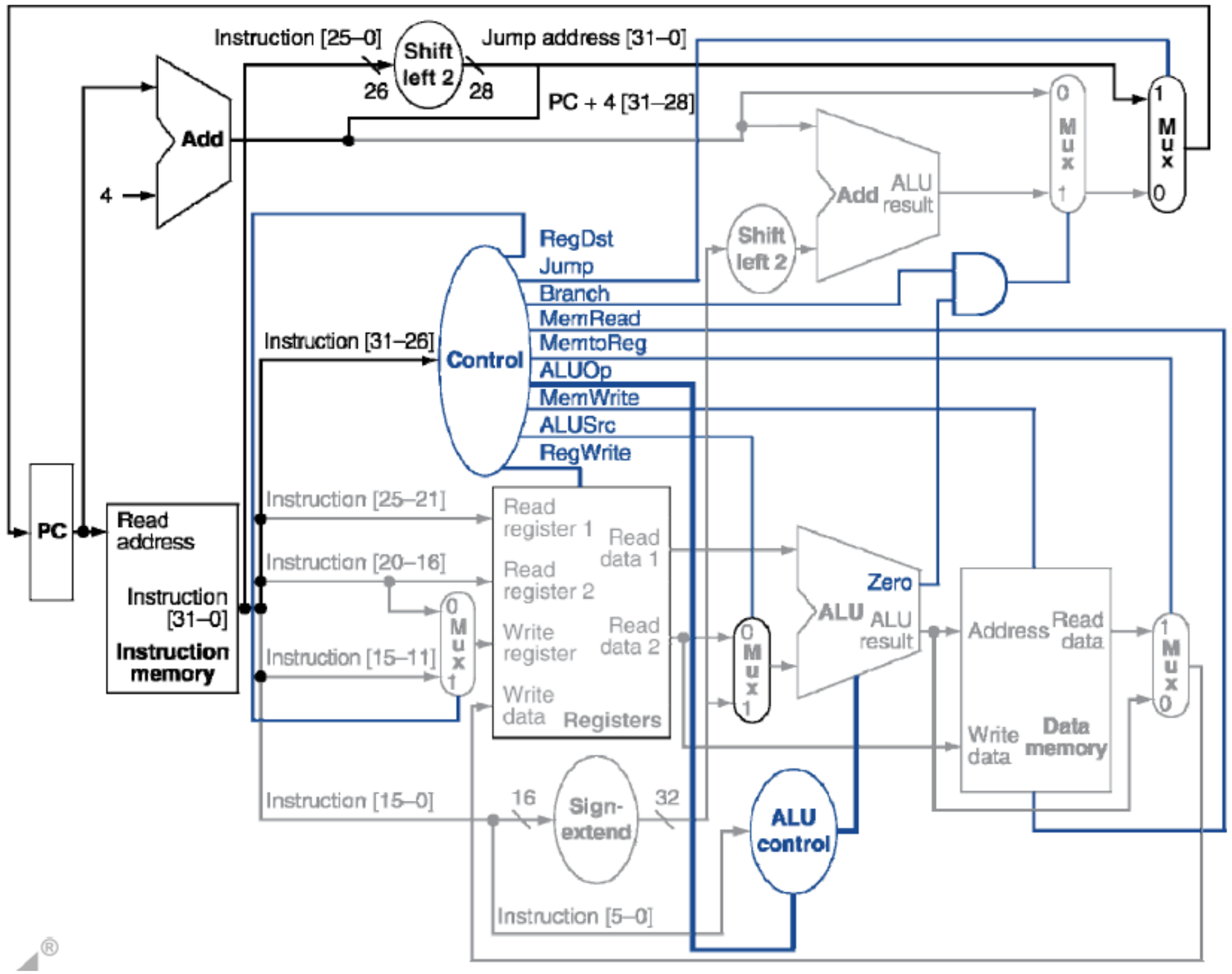





| | R-type | lw | sw | beq | jump |
|---|---|---|---|---|---|
| **Opcode** | 00 0000 | 10 0011 | 10 1011 | 00 0100 | 00 0010 |
| RegDst | 1 | 0 | X | X | X |
| Jump | 0 | 0 | 0 | 0 | 1 |
| Branch | 0 | 0 | 0 | 1 | X |
| MemRead | 0 | 1 | 0 | 0 | 0 |
| MemtoReg | 0 | 1 | X | X | X |
| MemWrite | 0 | 0 | 1 | 0 | 0 |
| ALUSrc | 0 | 1 | 1 | 0 | X |
| RegWrite | 1 | 1 | 0 | 0 | 0 |

(2) We wish to add the instructions subi (sub immediate) to the single-cycle machine. What changes do we need to make to the **control signals** and the **datapath**? Revise on the fig below, and modify the control signal table above.






| Opcode | Unique |
|---|---|
| RegDst | 0 |
| Jump | 0 |
| Branch | 0 |
| MemRead | 0 |
| MemtoReg | 0 |
| ALUop | sub |
| MemWrite | 0 |
| ALUSrc | 1 |
| RegWrite | 1 |